GriT: A CORBA-based GRID Middleware Architecture

Aniruddha S. Gokhale and Balachandran Natarajan
Institute for Software Integrated Systems
Vanderbilt University
2015 Terrace Place
Nashville, TN 37203, USA
{a.gokhale,b.natarajan}@vanderbilt.edu

Abstract

Rapid advances in networking, hardware, and middleware technologies are facilitating the development and deployment of Grid applications, which are characterized by their very high computing and resource requirements. These applications and services have multiple, simultaneous end-to-end quality of service (QoS) requirements, such as delay guarantees, jitter guarantees, security, scalability, reliability and availability guarantees, and bandwidth and throughput guarantees. Moreover, these applications and services require secure, controlled, reliable, and guaranteed access to different types of resources, such as network bandwidth, computing power, and storage capabilities, available from multiple service providers.

To support next-generation Grid applications effectively, there is a need to simplify grid programming by developing a new Grid middleware that raises the level of abstraction, and reduces the accidental complexities incurred by programming at the Grid infrastructure middleware level offered by existing Grid middleware such as Globus, ICENI, and Legion. Moreover, the new Grid middleware must ensure multiple end-to-end QoS properties simultaneously.

The paper provides three contributions to the research on next generation Grid middleware architecture that provides the above-mentioned properties. First, we describe how we are utilizing the standards-based CORBA distributed object computing and integration technology to design the next generation Grid middleware, called Grid TAO (GriT) that complements and enhances existing low-level Grid middleware, such as Globus. Second, we describe how we are using the real-time, fault-tolerant, and data parallel CORBA features to implement GriT to provide the desired properties. Finally, we show how CORBA's platform and language independence features are used in GriT to resolve the portability and interoperability challenges faced by current grid applications.

1 Introduction

1.1 Trends in Grid Computing

The emergence of next-generation distributed applications, such as *internet collaboration tools* for *telemedicine* and *scientific applications*, and *distributed mission training*, stems from a number of advances in networking, hardware, storage, middleware, and web technologies. For example, technologies such as *DiffServ*and Multi Protocol Label Switching (MPLS)are enabling service providers to provision and deliver network-level quality of service (QoS) to applications.

A special class of these distributed applications that have very high computing and resource requirements are called *Grid Applications*. Grid computing [7] is an emerging paradigm that seeks to harness the power of the internet and the sophisticated resources spread across it, such as super computers, storage devices, and others to support grid applications.

The grid computing paradigm envisages a distributed hardware infrastructure and a wide range of software infrastructure for services, programming models, tools, programming languages and methodologies capable of providing the massive computational requirements (Petaflops) and massive storage capacities (Petabytes) required by grid applications. Moreover, it is also expected to support high-fidelity, real-time collaboration between geographically distributed virtual organizations (VOs) [10], that comprise researchers, scientists, other users, and organizations.

A sampling of important grid application domains include:

- computationally intensive large-scale analysis of data such as seismic data, bioinformatics, satellite imagery of astronomical objects or surveillance data pertaining to enemy territory, and data resulting from experiments in elementary particle physics.

- computationally intensive high-fidelity realistic modeling and simulation of natural phenomena such as global climate change and mathematical ecology, and man-made systems such as groundwater transportation and traffic distribution.

The expected grid users include but are not limited to:

- government – for macro-level defense and disaster planning using the collective power of fastest computers
- astrophysicists – collaborating, collecting and processing data on the fastest computers
- bioinformaticians – analyzing human genome data
- meteorologists – studying earth atmospheric and climatic changes
- medical administrators and personnel – sharing specialized instruments like MRI machines and CAT scanners in addition to medical data and images
- economists and market analysts – studying and analyzing financial markets data

For grid applications to operate effectively, they simultaneously require:

- secure and controlled access to many different resources available from multiple resource and service providers, including networking resources (such as bandwidth and buffers), operating system resources (such as threads, CPU and kernel buffers), storage resources (such as high performance databases and RAIDs), computing resources (such as supercomputers), display resources capable of 3-D rendering, and many other specialized types of resources, such as sensors, telescopes, oscilloscopes, and other electronic equipment
- end-to-end multiple QoS properties, such as *delay guarantees*, *jitter guarantees*, *security*, *scalability*, *high reliability and availability guarantees*, and *bandwidth and throughput guarantees* to grid applications.

Developing these grid applications from scratch is not advisable and feasible due to the large number of inherent and accidental complexities arising from their stringent resource and end-to-end QoS requirements. The inherent complexities include dealing with multiple QoS properties of which some could cross-cut each other *e.g.,* tradeoffs between real-time and fault-tolerance, programming environments and paradigms totally different from some commonly used ones, sudden disruptions in service levels, *e.g.*, due to network partitioning or power outages. The accidental complexities include memory management differences across platforms, methodologies that are supported and efficient on one platform but not supported on the other, and errors that arise from improper handling of boundary conditions.

A solution to alleviate these inherent and accidental complexities is to implement a grid *infrastructure middleware* that can host grid applications and provide them with the required resource and QoS guarantees. Middleware is reusable software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware [23]. Its primary role is to bridge the gap between application programs and the lower-level hardware and software infrastructure to coordinate how parts of applications are connected and how they interoperate.

The *infrastructure middleware* that hosts grid applications is called a *Computational Grid* or simply a *Grid* [7]. Examples of infrastructure middleware include Globus [6], Legion [13] and ICENI [11] among others. The Grid provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities useful for *distributed super-computing*, *on-demand computing*, *high-throughput computing*, *data-intensive computing* and *collaborative computing*.

1.2 Middleware challenges supporting next-generation grid applications

Though existing infrastructure grid middleware seems suited to build next-generation grid applications, however, developing distributed grid applications using these is fraught with the following challenges. Moreover, unless several enhancements are made, conventional commercial off-the-shelf (COTS) *distribution middleware*, such as Common Object Request Broker Architecture (CORBA) [21] or J2EE [28], cannot host next-generation grid applications, although they help raise the level of abstraction and simplify distributed programming by eliminating many accidental complexities.

Challenge 1: Proliferation of grid infrastructure middleware choices: The proliferation of multiple grid infrastructure middleware technology platforms, such as Globus, Legion, or ICENI has raised the level of accidental complexity by increasing the amount of effort required to interoperate and port applications between Grid middleware technologies.

Moreover, service providers often have a considerable investment in legacy provisioning systems that do not even use today's Grid technologies. In particular, large-scale service provisioning systems often consist of components based on multiple software technologies developed over many years. Problems arise when these components must interoperate or be integrated together into a single executable since it is hard to assemble semantically compatible and interoperable components based on multiple middleware platforms.

With an increasing trend towards availability of newer middleware technologies, it is imperative for service

providers to interoperate seamlessly with emerging technologies without affecting existing grid applications.

Challenge 2: Satisfying multiple quality of service requirements simultaneously: As noted earlier, grid applications demand varying degrees and forms of QoS support from their middleware. For example, collaborative scientific applications involving geographically dispersed scientists, engineers, and physicists working on real-time experiments and data require the infrastructure to be efficient, predictable, scalable, secure, and fault tolerant. Owing to the complex nature of these QoS requirements, it is not feasible for a single vendor or product to develop an end-to-end solution that addresses all these challenges. Instead, highly configurable, flexible, and optimized COTS and non-COTS components from several different middleware providers based on *standards-based middleware* must be used to assemble and deploy these systems.

Challenge 3: Conventional COTS distribution middleware is too restrictive: Conventional COTS *distribution middleware*, such as CORBA or J2EE, is well-suited for only certain types of distributed applications, such as desktop and enterprise applications. Moreover, COTS middleware, in its current form, provides policies to manage resources only within the organization where it is deployed. Grid applications, however, manage and provision a diverse set of resources belonging to multiple service providers. This requirement entails the need to cross-cut service provider boundaries – a property lacked by conventional COTS middleware.

Challenge 4: Single sign-on secure access to resources with existing grid middleware is hard: As noted earlier, grid applications require simultaneous access to several different types of resources available from multiple resource and service providers that own them. These service providers include internet service providers (ISPs), storage service providers (SSPs), content service providers (CSPs), application service providers (ASPs), web hosting service providers (WHSPs) and others. For example, a distributed virtual surgery application involving geographically dispersed doctors, radiologists, medical professionals, and medical students will require high bandwidth for collaboration, large storage databases to hold patient records and radiology images, expensive display devices for precise 3-D modeling and rendering of images, virtual reality equipment for simulating surgeries, and telephony equipment to maintain multi-leg call sessions.

Applications that require these resources must maintain service level agreements (SLAs) with each individual service provider that provides the resources and services. Moreover, today's applications must authenticate themselves with each service provider everytime they access resources owned by the provider. Conventional COTS middleware is capable of provisioning only those resources and services belonging to the service provider where that middleware is deployed. It lacks the qualities to manage resources that involves cross-cutting service provider boundaries, however, making it infeasible for applications to have a *single sign-on* interface to access multiple resources and a common SLA applicable to all service providers. Existing grid infrastructure middleware does not address this problem either.

Addressing the challenges described above requires a new set of Grid *distribution middleware* services and components that complement and enhance existing grid infrastructure middleware while also raise the level of abstraction and simplifying distributed grid application development. Moreover, the new Grid distribution middleware must simultaneously be able to provide the following capabilities to grid applications:

- delivery of multiple end-to-end QoS properties,
- controlled and secure sharing of resources belonging to multiple service providers without the need for multiple SLAs and multiple authentications per session,
- portability across different low-level grid infrastructure middleware, and
- seamless interoperability between grid applications.

This paper describes a new Grid distribution middleware we are implementing, called *Grid TAO (GriT)*, that enhances standards-based CORBA middleware capabilities to complement and enhance earlier grid infrastructure middleware work, such as Globus [6] and Legion [13], to address the challenges outlined above.

The rest of the paper is organized as follows: Section 2 provides detail explanation of the GriT middleware architecture and shows how it addresses the challenges raised above; Section 3 describes related research; and finally Section 4 provides concluding remarks.

2 The Grid TAO (GriT) Middleware Architecture

This section describes our next generation Grid component middleware called Grid TAO (GriT). GriT enhances the Component Integrated ACE ORB (CIAO) [29] middleware, which is our CORBA Component Model (CCM) [19] implementation of the *The ACE ORB* (TAO) [25]. TAO is an open-source, high-performance, highly configurable CORBA Object Request Broker (ORB) that implements key patterns [26] to meet the demanding QoS requirements of distributed systems.

The following sections describe the various components of the GriT middleware architecture. We show how these components help resolve the challenges described in Section 1.2. We also illustrate the inherent technical challenges

faced in implementing GriT and provide our solutions to these.

2.1 Components of the GriT Middleware Architecture

Figure 1 illustrates the components of the GriT middleware architecture. As noted earlier, GriT is based on the standards-based CORBA technology.

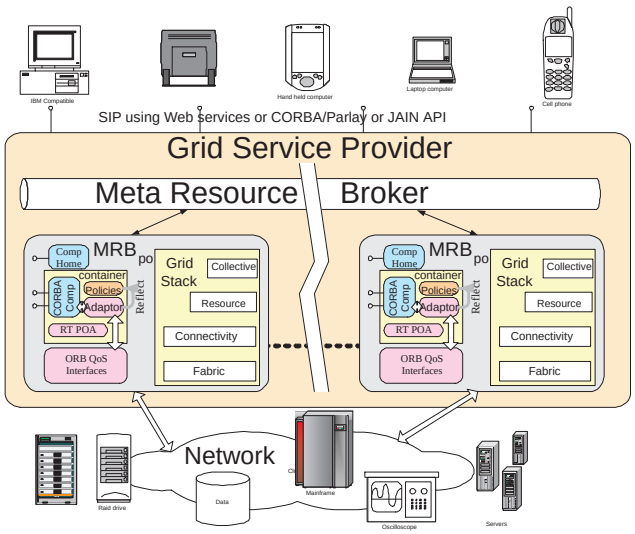

Figure 1. Grid TAO (GriT) Middleware Architecture

Below we explain each component of the architecture in detail.

2.1.1 Grid Service Provider (GSP)

The GriT middleware comprises the notion of a Grid Service Provider (GSP) similar to other service providers such as ISPs, SSPs, ASPs, WHSPs, CSPs and other providers providing specialized services such as access to advanced displays, virtual reality equipment, telescopes, oscilloscopes, etc. A fundamental difference between a GSP and other service providers is that a GSP is an abstract notion and does not actually own resources. The goal of the GSP is to provide a standard, unified view of resources to grid applications thereby eliminating the need for grid applications to require the knowledge and location of individual resource service providers. Figure 2 illustrates the concept of a GSP.

The GSP provides a *single sign-on* capability for grid applications, which eliminates the need for multiple SLAs and authentication mechanisms with multiple service providers. Applications use the GSP to delegate the responsibility of authenticating themselves with the individual specialized service providers. Moreover, the GSP offers its user interface as a standard web service, thereby enabling grid

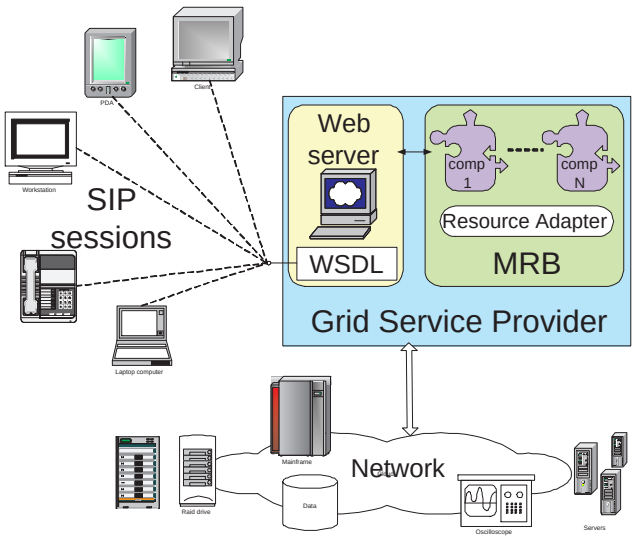

Figure 2. Overview of Grid Service Provider (GSP)

clients to use techniques such as session initiation protocol (SIP) [2] to create, join, or leave collaborative grid applications.

2.1.2 Meta Resource Broker (MRB)

At the heart of the GSP is a Meta Resource Broker (MRB), which is an enhanced CORBA ORB, that encapsulates resources from multiple providers as CORBA objects. The MRB exemplifies the actual GriT middleware. The MRB provides applications with standards-based, uniform interfaces and mechanisms to access and manage the underlying resources, and to create or join new or existing collaborative sessions, respectively. Figure 3 illustrates the MRB concept.

The MRB mediates requests for different services and resources on behalf of grid applications and delivers them with the resources and guaranteed QoS. This is accomplished by the MRB delegating the task of looking up individual resources required by the grid applications to MRB *part objects*. Figure 3 illustrates the concept of a primary MRB parallel object and its part objects. These reusable patterns are defined in the Data Parallel CORBA (DP-CORBA) [20] specification.

The MRB, by way of delegating responsibility to its part objects, reserves and manages different types of *virtual* resources, which are high-level abstractions of resources, such as network bandwidth, databases, or supercomputers, belonging to different service providers. The resources are virtual since the GSP does not actually own any resources, but maintains only abstractions of them.

When a grid application makes a reservation request to

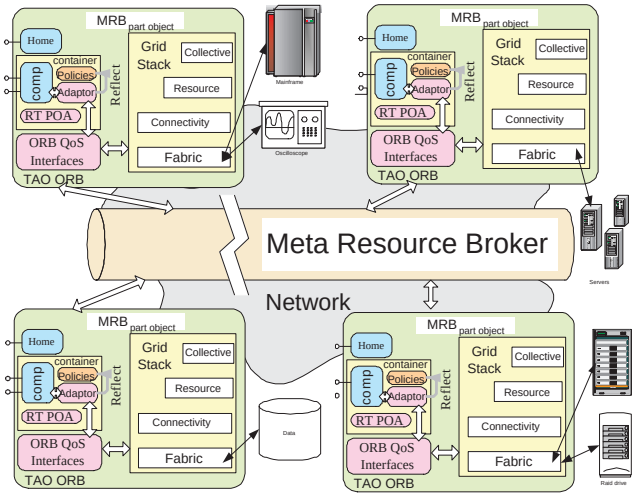

Figure 3. Meta-Resource Broker Architecture

the GSP for all the different resources it needs and the QoS guarantees, this request is handed down by the GSP to its underlying MRB parallel object. The MRB parallel object will in turn partition the request using the techniques described in the DP-CORBA specification and the *Data Reorganization Effort* (`www.data-re.org`), such as *block distribution* or *cyclic distribution*. The partitioned request is then handed over to the MRB part objects. Each MRB part object is responsible to discover the appropriate resources that can meet the application's QoS requirements. This discovery process is performed in parallel thereby providing a highly scalable and predictable solution to determine the feasibility of resource and service provisioning.

The mechanism of resource discovery outlined above is akin to a *nested transaction*. If any one of the *child* transaction *i.e.,* resource discovery undertaken by a part object, is unsuccessful, then the *parent* transaction *i.e.,* the request initiated by the MRB parallel object, is rolled back.

If the request for resources is feasible, then the result of the MRB part object resource discovery operation is a collection of resources required by the application that provides it with the QoS guarantees. This collection of virtual resources is subsequently managed by the MRB as another *parallel object*. It is then upto the grid application to efficiently utilize these resources.

Each MRB part object is implemented using the CORBA Component Model (CCM), where components encapsulate the logic and policies to manage the *virtual* resources. Since these components manage virtual resources, they serve as a resource *proxy* of the actual resources thereby providing a uniform view to client applications. Figure 4 illustrates the architecture of a MRB part object illustrating the use of CORBA components and an ORB enabled for parallel computing and other QoS requirements, such as real-time and fault tolerance. As noted earlier, the ORB and the component/containers functionality in the MRB is provided by CIAO, which is our CCM implementation of the TAO ORB.

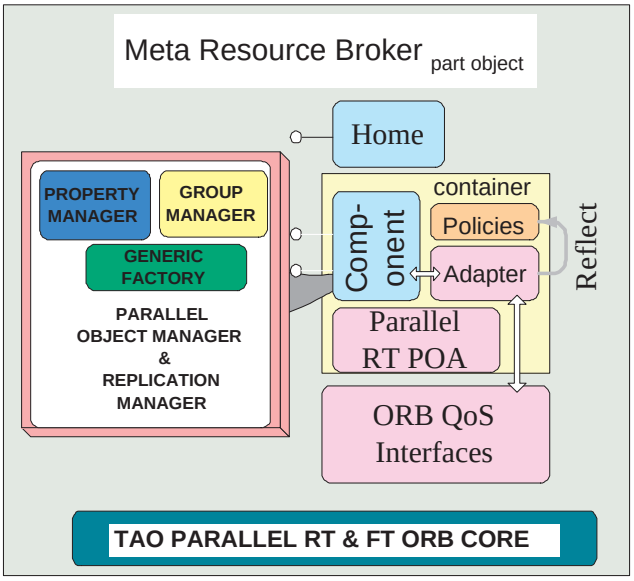

Figure 4. Meta-Resource Broker Part Object Architecture

2.2 Resolving Grid Application Challenges with GriT

As described in Section 1.2, developing grid applications using today's Grid middleware technologies requires grid application developers to address the following challenges:

- dealing with proliferation of low level grid infrastructure middleware technologies
- satisfying multiple quality of service requirements simultaneously
- restrictions using conventional COTS middleware
- complexity in provisioning single sign-on secure access to resources with existing COTS middleware

Below we explain how GriT addresses the challenges outlined above.

2.2.1 Challenge 1: Proliferation of low-level Grid middleware choices

Context The proliferation of multiple low-level Grid middleware technology platforms, such as Globus, Legion, or ICENI has raised the level of accidental complexity by increasing the amount of effort required to interoperate and port applications between Grid middleware technologies.

Solution As mentioned before, GriT uses standards-based, platform and language independent CORBA integration and distribution technology. This raises the level

of abstraction and simplifies distributed grid programming. Moreover, GriT's GSP offers a standards-based web service interface to grid applications. This enables grid applications to seamlessly interoperate between different underlying low-level grid middleware technologies. Furthermore, standards-based interfaces ensures application portability.

2.2.2 Challenge 2: Satisfying multiple quality of service requirements simultaneously

Context Grid applications are characterized by their demand for varying degrees and forms of QoS support from their middleware.

Solution As noted earlier, GriT is being built using the high-performance, real-time CORBA-compliant ORB called TAO [15], which is a open-source middleware developed as part of our previous and ongoing research activites. The MRB delegates resource lookup and application authentication to the part objects. The part objects in turn return the resources that best meet the needs of the applications. Moreover, the real-time and fault-tolerance features of the TAO ORB ensure guaranteed end-to-end grid application request priorities and reliability, respectively. COTS middleware, such as CORBA, have advanced to a level of enabling applications to deliver real-time QoS requirements [16, 24].

2.2.3 Challenge 3: Conventional COTS middleware is too restrictive

Context Grid middleware must manage and provision a diverse set of resources belonging to multiple service providers. This requirement entails the need to cross-cut service provider boundaries – a property lacked by conventional COTS middleware.

Solution As explained before, the MRB is implemented as a parallel CORBA object. The MRB's part objects are deployed on individual service provider infrastrutures after maintaining the proper SLAs with each of them. However, these SLAs are transparent to the grid applications who are required to only maintain a single SLA with the GSP. This arrangement allows GriT to cross-cut service provider boundaries.

2.2.4 Challenge 4: Single sign-on secure access to resources with existing COTS middleware is hard

Context Grid applications require simultaneous access to several different types of resources available from multiple resource and service providers that own them. This implies having to maintain SLAs and authenticating with each individual service provider that provide the resources and services. Conventional COTS middleware lacks the qualities to manage resources that involves cross-cutting service provider boundaries, however, making it infeasible for applications to have a *single sign-on* interface to access multiple resources and a common SLA applicable to all service providers.

Solution As noted in the solution to challenge 3 above, GriT is able to cross-cut service provider boundaries by maintaining individual SLAs with them thereby enabling GriT to access the resources via the individually deployed MRB part objects. The grid applications, on the other hand, are shielded from the underlying service providers by the GSP. Applications need to maintain only a single SLA with the GSP. Any request for resources is handled by the GSP on behalf of the application.

2.3 Addressing Technical Challenges Implementing GriT

This section describes the technical challenges that must be addressed by Grid middleware like GriT.

2.3.1 Avoiding Starvation of Transient Collaborative Applications

Context. Applications collaborating in an *ad hoc* peer-to-peer manner will require resources to be reserved and provisioned for only the duration of the collaboration. For example, a group of scientists might decide to setup a collaborative conference to discuss certain aspects of a project. Resources will therefore need to be provisioned for the duration of the conference only after which the resources are available for use in another Grid application.

On the other hand, long running collaborative scientific experiments might need to be stopped from time-to-time to analyze intermediate results and resumed from the point where the application was previously stopped. This would require that resources be provisioned for the entire duration of the experiment irrespective of the temporary inactivity.

Problem. Existing Grid resource management techniques allow resource reservation from multiple providers without making any distinction between transient and persistent collaborations. Moreover, they do not optimize resource usage whenever persistent collaborative applications have temporarily stopped utilizing the resources. If several persistent applications reserve resources for long durations of time, it might starve other transient applications.

Solution. GriT enhances the CORBA Portable Object Adapter (POA) *lifespan policies* to incorporate the transient and persistent nature of collaborative applications. In particular, for the persistent collaborative applications, the MRB's POA marks the resource as being persistent. This way, the next time the application decides to resume the session, the MRB will figure out the QoS and resource requirements of the application and provision the same set of resources. Moreover, when the application has temporarily stopped using the resource, the MRB will mark the resource

as temporarily available thereby allowing it to be used for a short duration transient collaborative application. This optimization ensures that resource utilization is optimized and transient collaborative applications do not starve for resources.

2.3.2 Dispatching Requests to Concrete Resources Transparently

Context. An incoming client request is demultiplexed and dispatched by the ORB and the POA to servants within the same address space as the POA. In MRB, however, an incoming request for a resource gets demultiplexed and dispatched to a servant that represents a virtual resource.

Problem. In MRB, the servant that handles incoming requests for resources must transparently forward the request to the concrete resource corresponding to the virtual resource.

Solution. The MRB POA's *active object map*, which keeps an association between an object reference and its associated servant, is modified to also maintain a mapping between the virtual resource and the actual resource it represents. This enables any incoming client request to be forwarded transparently to the concrete resource over the appropriate communication link.

2.3.3 Alleviating Resource Redundancy

Context. Dependability is a key QoS requirement for Grid applications. This includes reliability and high availability of resources. This becomes particularly important for applications such as virtual surgery or mission training.

Problem. To guarantee reliability and high availability, the MRB must provision multiple number of redundant resources. This implies that a single virtual resource must now represent a collection of redundant resources. This impacts the mapping of a virtual resource into the actual resource since now we need a policy for choosing one of the concrete redundant resources. Furthermore, we need an addressing mechanism to address a group of redundant resources.

Solution. To provide high degree of reliability and availability to Grid applications, the component representing the virtual resource maintained by the MRB actually represents a set of redundant resources that the MRB has brokered as part of the application's QoS specifications and SLA. The choice of which resource is selected from among the group of redundant resources is configurable into the MRB dynamically. For example, one policy might use the resources in a round-robin manner, while another policy might require sophisticated load-balancing algorithms to determine the choice of the resource selected. For this, we are using the *Abstract Factory*, *Strategy* [12], and Component Configurator [26] patterns in the MRB's POA to configure the concrete resource selection policy.

An additional challenge is in addressing the group of redundant resources. We are using a multiple profile interoperable component reference (ICR) to address the group, where each profile addresses an individual resource in the group. To support ICR, modifications to the TAO ORB core have been made.

2.3.4 Interacting with existing Grid Protocols

Context. The Grid protocol hierarchy comprises four layers [10] illustrated in Figure 5. These four layers include: (1) a Fabric layer – which provides the resources and the protocols to access the resources, (2) a Connectivity layer – which provides the core communication and authentication protocols for exchanging Fabric layer data, (3) the Resource layer – which provides the protocols to securely access, monitor, control, and meter the usage of individual resources, and (4) the Collective layer – which comprises protocols to capture interactions amongst resources at a global level.

Problem. The Grid computing model [10] imposes the four-layer protocol stack for Grid applications to use. However, there is not yet a standardization of the protocols at all the four layers. Different Grid middleware frameworks will therefore have their own set of protocols at the four layers. Thus, there is a need for inter-Grid protocols to achieve interoperability. Moreover, many Grid developers today use lower-level Grid protocol APIs to program their applications, which is tedious and error-prone due to the lack of a higher-level type system that can be checked by compilers.

Solution. As mentioned earlier, GriT complements and enhances existing Grid infrastructure middleware protocols. Moreover, GriT shields grid applications from the differences in the protocols provided by different Grid frameworks by providing a standards-based CORBA and web services API. Figure 5 illustrates the interactions of the MRB part object, which is deployed at the actual service provider's premises after proper authorization, with different protocols of the Grid protocol stack. To achieve inter-Grid interoperability or processing Grid-specific application requests, GriT uses the *Pluggable Protocol* [22] framework that adapts higher-level CORBA requests to the underlying Grid protocol messages.

In the following, we describe how the MRB interacts with the Grid protocols offered by the Globus [6] toolkit, which is an instance of a grid infrastructure middleware. We also describe any enhancements we make to existing grid protocols.

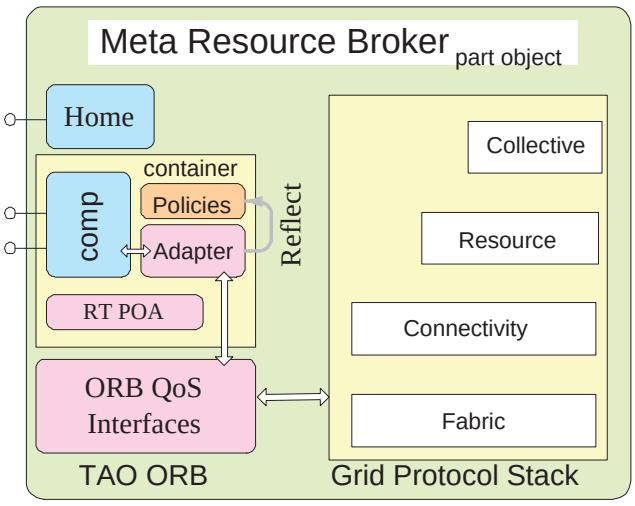

Figure 5. Interactions of the Meta Resource Broker Part Object with Grid Protocols

Fabric layer GriT reuses the functionality provided by the Globus's General-purpose Architecture for Reservation and Allocation (GARA) [8] to actually perform the resource reservations. The MRB transforms incoming client CORBA IIOP or HTTP requests for resource reservations into GARA requests. This is done transparently to the client applications. A challenge here is to map CORBA IDL or Web Services Description Language (WSDL)operations invoked by Grid applications into appropriate GARA requests.

Connectivity layer As noted earlier, the primary goal of the GSP is to provide a single sign-on capability for client applications. These client sign-on requests serve to delegate the responsibility of authentication onto the GSP. In the delegation phase, the MRB uses the Grid Security Infrastructure (GSI) [14] protocol requests that will take care of authenticating the GSP, and hence the client. Since the GSP acts on behalf of the application, the GSP itself maintains multiple SLAs with different service providers. The challenge here is for the MRB to predictably and scalably map incoming application authentication requests into GSI requests to those providers with whom the GSP's SLA offers the desired service.

Resource layer For resource management and accounting purposes, the MRB's resource management and accounting system reuses the functionality offered by the Grid Resource Allocation Manager (GRAM) [3], and other LDAP-based protocols implemented in Globus.

Collective layer For resource discovery, the MRB utilizes Globus's Meta Directory Service (MDS) [4] protocol. We are exploring ways to combine the CORBA Trading service and Uniform Description, Discovery, and Integration (UDDI) with MDS. For fault-tolerance, we have extended the fault-tolerant CORBA standard's *infrastructure-controlled membership* policy to use Globus's dataset and resource replication strategies. Moreover, the Heartbeat Monitor (HBM) fault detection protocols in Globus are used to detect resource failures. To ensure scalability, we are using the fault-tolerant CORBA's concept of *domain* and arranging the HBMs in a hierarchical structure.

2.3.5 Simplifying the programmming interface to Grid Service Provider

Context. Both wireless and wireline client applications must be able to participate in collaborative Grid applications. This requires *thin* client applications that can use the GSP's interface to share resources.

Problem. Programming directly at the Grid framework-specific protocols is too low-level and hence tedious and error-prone. Moreover, for small footprint wireless clients and other embedded devices to use the Grid framework, standards-based protocols and interfaces must be used.

Solution. The services offered by the GSP are hosted as a web service as shown in Figure 2. This approach is similar to the ideas proposed in the Open Grid Services Architecture (OGSA) [9], where the GSP's services are described in the Web Service Description Language (WSDL). Client applications access the GSP services via the web by contacting the UDDI registry.

This feature provides Grid application developers tremendous benefits when establishing SIP sessions. SIP [2] is designed to enable two or more participants to establish a session consisting of multiple media streams including audio, video, and other internet-based communication mechanisms such as distributed gaming, shared applications, whiteboards, etc. Participants in a collaborative application use the GSP's interfaces and services to set up SIP-enabled collaborative sessions. As mentioned earlier, these sessions could be set up in an *ad hoc* manner or they can be persistent sessions.

3 Related Work

3.1 Related work on middleware

Popular COTS component middleware platforms being used for various distributed applications today include the CORBA Component Model (CCM) [19], J2EE [28], and COM [1]. CCM is modeled closely on the Enterprise Java Beans (EJB) specification. Unlike EJB, however, CCM uses the CORBA object model as its underlying object interoperability architecture and is therefore not bound to a particular programming language. CCM and CORBA are also related

to the Microsoft COM family of middleware technologies. Unlike CORBA, however, Microsoft's COM was designed to support a collocated component programming model initially and later DCOM added the ability to distribute COM objects.

We base our work on the CCM since CORBA is the only standards-based COTS middleware that has made a substantial progress in satisfying the QoS requirements of distributed systems. For instance, the real-time, fault-tolerant, and data parallel CORBA specifications have been adopted in recent years.

The GriT middleware described in this paper uses our CCM implementation called CIAO [29]. CIAO enhances our earlier TAO ORB to simplify the development of QoS-enabled distributed applications by enabling developers to statically provision QoS policies end-to-end declaratively when assembling a system.

3.2 Related work on Grid Computing

Grid computing is an emerging powerful paradigm to build large-scale, distributed, collaborative applications that require secure, controlled access to different resources from multiple providers.

The National Science Foundation (NSF)'s Next-generation Middleware Initiative (NMI) program has led to the formation of a GRIDS center (`http://www.grids-center.org/`), where several innovative ideas in Grid Computing [10] are being deployed to mature the Grid.

The Globus toolkit [6] (`http://www.globus.org`) is well established Grid middleware that serves as a networked virtual supercomputer or a metacomputer and uses high-speed networks to connect supercomputers, databases, scientific instruments, and advanced display devices, possibly geographically dispersed.

The Globus toolkit, considered the reference Grid implementation, is an open source, open architecture framework that provides a suite of tools to enable Grid application developers to build computational grids and grid-based applications.

Legion [13] is an object-based metasystems Grid middleware designed for a system of millions of hosts and trillions of objects tied together with high-speed links. Users of Legion get the illusion of a single computer with access to all kinds of data and physical resources.

ICENI [11] is a component-based Grid middleware implemented in Java. It uses the Jini technology to lookup resources. ICENI supports management of collection of computational resources and provides scalable secure access to authorized users.

The GriT middleware described in this paper is a distribution middleware that complements and enhances the low-level grid infrastructure middleware such as Globus, Legion, and ICENI.

The GSI [14, 5] protocol used in Globus enables secure authentication and communication over an open network. The primary advantage of GSI is its ability to support mutual authentication and single sign-on. In our work on the MRB, we extend the GSI capabilities to provide authorization and encryption by integrating it with the CORBA security service [18]. For example, certain collaborative applications involving classified projects might require that participants be authorized to use the application. Moreover, all authorized collaboration might need to be encrypted.

The Globus MDS [4] provides the tools to build an LDAP-based information infrastructure for computational grids. By using standards-based protocols, such as LDAP, MDS provides a uniform means of querying system information from a rich variety of system components. Since GriT uses CORBA and provides a web service user interface, we have integrated MDS with the CORBA Trader Service and web service UDDI. This integration enables a large number of applications to use standard interfaces for discovering resources. Moreover, the Trader Service offers a very rich lookup capability that can be used to select resources that fit best for a Grid application.

The Globus resource management architecture is a layered system in which high-level global resource management services are layered on top of local resource allocation services. The GRAM [3] serves as a broker for requesting and acquiring resources. Since GRAM is a Resource layer protocol, the Grid architecture [10] recommends having a very small subset of protocols at this layer. As a result, we are reusing the capabilities of GRAM. The MRB uses the pluggable protocol framework [22] in the TAO [15] ORB to reuse the GRAM protocol.

The fault detection service in Globus is the Heartbeat Monitor [27], which enables a process to be monitored and periodic heartbeats to be sent to one or more monitors. We have extended the HBM functionality by a much richer, high performance, fault tolerant CORBA [17] we developed that provides fault tolerance at the object level. The finer granularity is required since resources within the MRB are represented as CORBA objects. Moreover, high performance is essential since failure recovery times must be predictable to support end-to-end QoS.

4 Conclusions

This paper describes a next generation Grid middleware called GriT. GriT is designed to complement and enhance existing low-level Grid middlware such as Globus.

The key ideas in GriT include the Grid Service Provider (GSP) and the Meta-Resource Broker (MRB). The GSP offers a web service interface to grid applications thereby sim-

plifying grid programming. Moreover, the GSP eliminates the need for grid users to maintain multiple service level agreements (SLAs) with individual service providers. The meta resource broker (MRB) manages virtual resources that are abstractions of concrete resources belonging to multiple service providers.

References

[1] D. Box. *Essential COM*. Addison-Wesley, Reading, MA, 1998.

[2] U. S. Corporation. White Paper: SIP and SOAP. `www.sipforum.org/whitepapers/USC-SIPSOAP-WP2.pdf`.

[3] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer-Verlag LNCS 1459, 1998.

[4] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 365–375. IEEE Computer Society Press, 1997.

[5] I. Foster, N. Karonis, C. Kesselman, G. Koenig, and S. Tuecke. A Secure Communications Infrastructure for High-Performance Distributed Computing. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 125–136. IEEE Computer Society Press, 1997.

[6] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[7] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Harper Collins, 1999.

[8] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service (IWQOS'99)*, London, UK, May 1999.

[9] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. `www.globus.org/research/papers/ogsa.pdf`, Jan. 2002. DRAFT.

[10] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3):205–220, Apr. 2001.

[11] N. Furmento, A. Mayer, S. Gough, S. Newhouse, T. Field, and J. Darlington. An integrated grid environment for component applications. In *Proceedings of the Second International Workshop on Grid Computing- Grid 2001, Denver 2001*. Springer-Verlag LNCS, 2001.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[13] A. S. Grimshaw and W. A. W. et al. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, Jan. 1997.

[14] I. Foster and C. Kesselman and G. Tsudik and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computers and Security*, pages 83–91. ACM Press, 1998.

[15] Institute for Software Integrated Systems. The ACE ORB (TAO). www.dre.vanderbilt.edu/TAO/, Vanderbilt University.

[16] R. Lachenmaier. Open Systems Architecture Puts Six Bombs on Target. `www.cs.wustl.edu/~schmidt/TAO-boeing.html`, Dec. 1998.

[17] B. Natarajan, A. Gokhale, D. C. Schmidt, and S. Yajnik. Applying Patterns to Improve the Performance of Fault-Tolerant CORBA. In *Proceedings of the 7^{th} International Conference on High Performance Computing (HiPC 2000)*, Bangalore, India, Dec. 2000. ACM/IEEE.

[18] Object Management Group. *Security Service Specification*, OMG Document ptc/98-12-03 edition, Dec. 1998.

[19] Object Management Group. *CORBA 3.0 New Components Chapters*, OMG TC Document ptc/2001-11-03 edition, Nov. 2001.

[20] Object Management Group. *Data Parallel CORBA Specification*, ptc/2001-11-09 edition, Nov. 2001.

[21] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.6*, Dec. 2001.

[22] C. O'Ryan, F. Kuhns, D. C. Schmidt, and J. Parsons. Applying Patterns to Develop a Pluggable Protocols Framework for ORB Middleware. In L. Rising, editor, *Design Patterns in Communications*. Cambridge University Press, 2000.

[23] R. E. Schantz and D. C. Schmidt. Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications. In J. Marciniak and G. Telecki, editors, *Encyclopedia of Software Engineering*. Wiley & Sons, New York, 2001.

[24] D. C. Schmidt. R&D Advances in Middleware for Distributed, Real-time, and Embedded Systems. *Communications of the ACM special issue on Middleware*, 45(6):43–48, June 2002.

[25] D. C. Schmidt, B. Natarajan, A. Gokhale, N. Wang, and C. Gill. TAO: A Pattern-Oriented Object Request Broker for Distributed Real-time and Embedded Systems. *IEEE Distributed Systems Online*, 3(2), Feb. 2002.

[26] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, 2000.

[27] P. Stelling, I. Foster, C. Kesselman, C.Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In *Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing*, pages 268–278. IEEE Press, 1998.

[28] Sun Microsystems. Java^{TM} 2 Platform Enterprise Edition. java.sun.com/j2ee/index.html, 2001.

[29] N. Wang, D. C. Schmidt, A. Gokhale, C. D. Gill, B. Natarajan, C. Rodrigues, J. P. Loyall, and R. E. Schantz. Total Quality of Service Provisioning in Middleware and Applications. *The Journal of Microprocessors and Microsystems*, 27(2):45–54, mar 2003.